Aging Assessment and Design Enhancement of Randomized Cache Memories

David Trilla, Carles Hernandez, Jaume Abella,and Francisco J. Cazorla

***Abstract*—Critical real-time systems require the estimation of the Worst-Case Execution Time (WCET) for scheduling purposes and resource budgeting. Measurement-Based Probabilistic Timing Analysis (MBPTA) has been shown recently as a powerful approach for WCET estimation. MBPTA builds upon time-randomized cache memories. While aging has been deeply analyzed for conventional time-deterministic caches (i.e. implementing modulo placement), little work has been done to assess the reliability of time-randomized caches. In this line, only the WCET analysis of faulty time-randomized caches has been introduced recently. However, the intrinsic robustness of randomized hardware designs has not been assessed yet. In this paper we perform, for the first time, an assessment of the aging-robustness of random placement cache designs: random modulo and hash-based random placement. We propose a new random modulo implementation preserving its key benefits in terms of low critical path impact, low miss rates and MBPTA compliance; while reducing hot-carrier injection aging by achieving a better – yet random – activity distribution across cache sets. On the other hand we show that gains in terms of BTI aging are limited for random placement designs on their own.**

***Index Terms*—Cache Memories, Random Placement, HCI, BTI, real-time systems, safety.**

I. Introduction

Critical real-time systems such as those in the avionics and automotive domains (among others) deal with functionalities with human in the loop or that relate to the integrity of the system itself. This requires assessing that those systems will perform their operation *correctly* and *in time* following the guidelines in the corresponding safety standards (i.e. ISO26262 in automotive [17] and DO-178B/C in avionics [26]).

Timing verification requires deriving Worst-Case Execution Time (WCET) of the programs analyzed for scheduling purposes. Recently, Measurement-Based Probabilistic Timing Analysis (MBPTA) [8] has been proposed to derive reliable WCET estimates on top of complex hardware. MBPTA benefits from hardware platforms providing some properties such as random placement and replacement in caches [19], [20], [15]. In this line, different implementations of random placement have shown to be suitable for first-level (L1) and second-level (L2) caches. In particular, hash Random Placement (*hRP*) [20] has been shown convenient for L2 caches whereas Random Modulo (*RM*) has been shown more convenient for L1 caches [15]. While they simplify the use of MBPTA to obtain probabilistic WCET (pWCET) estimates, their fault tolerance – needed for functional correctness – has been barely considered. In fact, the robustness properties of randomized hardware platforms have only been exploited to derive pWCET estimates that hold valid in the presence of faults in random placement and replacement caches [29], [30].

This paper makes a step forward in the reliability analysis of time-randomized caches with the following contributions: (1) We analyze the impact of *RM* in terms of hot carrier injection (HCI) aging and bias temperature instability (BTI) for L1 caches showing that benefits in terms of HCI are meaningful and can be further improved, whereas BTI gains are very limited. (2) We propose an enhanced *RM* design to mitigate HCI aging. Finally, (3) we analyze the impact of *hRP* in terms of HCI in L2 caches showing that benefits are already huge.

The rest of the paper is organized as follows. Section II provides background on random placement designs and aging. Section III presents the reliability assessment of *RM* and *hRP* and our enhanced *RM* design. Those different implementations are evaluated in Section IV. Section V describes some related work. Finally, Section VI concludes this work.

II. Background

In this section we first present *RM* and *hRP* cache designs, needed for MBPTA, as they are the basis for our work. Then, we briefly introduce HCI and BTI as exemplary sources of aging.

A. Random Placement Cache Designs

MBPTA is a timing analysis method that derives a pWCET estimate to bound the execution time of tasks. The application of MBPTA requires relating the execution time behavior observed during the analysis phase with that during operation. This requires, among other things, that the execution time jitter introduced by hardware components is controlled in some way [22]: (a) by exercising appropriate inputs during the analysis, (b) by enforcing the highest latency response time during the analysis, or (c) by enforcing a time-randomized behavior during both the analysis and operation.

Randomization has been shown particularly effective to remove the impact on cache placement of the potential memory placement experienced during operation. In other words, by randomizing cache placement, cache conflicts are independent of the actual addresses where code and data are placed as long as cache-line alignment can be preserved, which is regarded as doable in the critical real-time domain [19]. Cache memories are time-randomized by using random placement and replacement policies [19], [20], [15]. Those designs have

All authors are with the Barcelona Supercomputing Center (BSC). Barcelona, Spain. D. Trilla is also with the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. F.J. Cazorla is also with the Spanish National Research Council (IIIA-CSIC). Barcelona, Spain.

D. Trilla is the corresponding author. Address: c/ Jordi Girona, 29, 08034, Barcelona (Spain). Telf: +34 934137170. Fax: +34 934137721. Email: david.trilla@bsc.es

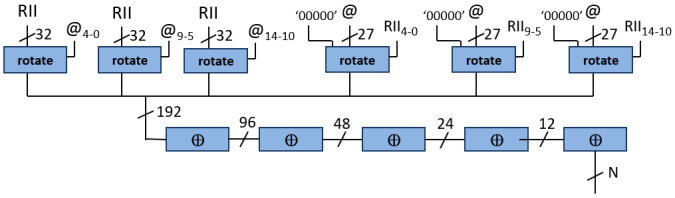

Fig. 1. Schematic of the hash logic of *hRP*.

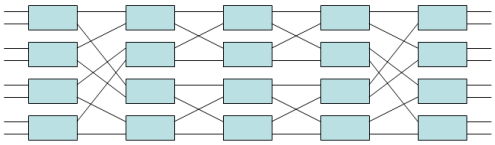

Fig. 2. Example of a 4-bit Benes network.

been implemented in a RTL prototype of a 4-core LEON3 processor from the Space domain [14]. In particular, *RM* has been implemented in L1 data and instruction caches and *hRP* in the L2 cache, which has been shown to be the most convenient configuration [15]. While performance on top of time-randomized caches cannot be regarded as higher or lower than that on top of conventional caches with modulo placement and LRU replacement in the general case [24], they have been shown to provide execution times less than 2% higher than those on conventional caches [15]. Note that conventional caches are intended for optimizing average performance while time-randomized ones are intended to provide low pWCET.

Hash-based random placement, *hRP*, maps memory addresses in the same cache-line boundary randomly to a cache set based on a random seed, which is changed across program runs. While this provides the properties needed by MBPTA, it may produce bad placements in terms of miss rates: even if a program accesses few cache lines, those lines may be randomly placed into the same cache set. *hRP* is implemented using a hash function that rotates address bits based on a random seed (RII) and RII bits based on some address bits. Finally, all bits of those rotations are XORed to obtain the cache set index, as shown in Figure 1.

Conversely, deterministic modulo (M) placement maps consecutive memory lines into consecutive sets, thus avoiding this type of conflicts. However, conflicts across lines are not random and strictly depend on memory location. Since memory location of objects during operation is hard to be controlled and mimicked at analysis time, conflicts at analysis are unlikely to represent those during operation, hence thwarting the use of MBPTA.

Instead, random modulo, *RM*, randomizes cache placement within cache way boundaries (also using a random seed) so that, as long as cache ways do not exceed memory page size (typical case for L1 caches), consecutive cache lines within cache way boundaries cannot conflict among them by construction, as it is the case for deterministic modulo placement. This is achieved by randomizing placement in the form of a random permutation. Still, conflicts among lines beyond cache way boundaries are random. Hence, average performance is close to that of deterministic modulo placement and worst-case placements deliver performance close to the average performance [15]. However, *RM* cannot be used in L2 caches since cache ways are much larger than page size, so *hRP* is the only choice for L2 caches.

Random modulo is implemented by means of a Benes network where each node allows input signals to go through or to commute based on control signals (see Figure 2). In the case of random modulo, input bits (those coming from the left in the figure) are those address bits used as index in regular modulo placement, and control bits (one control bit per box, not shown in the figure) are produced by XORing conveniently the tag bits[1] and a random seed that is changed across program executions. In this way a given memory line is randomly placed in cache, such placement holds constant during the whole execution, and it changes randomly across executions by changing the random seed (and flushing cache contents). Since addresses within cache way boundaries have distinct index bits, the Benes network delivers a bijective function so that, given specific control signals (those produced by addresses with the same tag with the same random seed) each index is placed in one set and each set corresponds to exactly one cache index. Therefore, a permutation is obtained and conflicts cannot occur across lines with identical tag.

Note, however, that the particular way to combine tag bits and the random seed determines the particular index bit permutation chosen. Also, the output of the network is determined by the particular index bits of the address being accessed. This is detailed in the next section, where the reliability assessment is performed. In particular we show how the particular address-to-set mapping influences the utilization of each set and, therefore, their degradation in terms of HCI.

B. HCI and BTI Aging

Next we briefly introduce two representative sources of aging and their relation with cache activity: HCI [32] and BTI [27]. HCI and BTI are particularly interesting because of their different relation to circuit activity.

HCI [32] aging occurs when a carrier (either an electron or a hole) is injected from the conducting channel into the silicon substrate or the gate dioxide, where it stays permanently trapped. Then, whenever the gate is intended to charge or discharge current, further electron-hole pairs need to be made due to the trapped carrier, thus affecting negatively both gate delay and leakage, and potentially making the gate fail its specifications. HCI, among other sources of transistor degradation, affects devices proportionally to the activity produced, which in turn depends on the access distribution across cache sets. Hence, we aim at finding a better random modulo implementation that balances utilization of the cache sets regardless of the particular access pattern and indexes of the addresses accessed.

BTI [27], instead, breaks progressively silicon-hydrogen bonds at the silicon/oxide interface whenever a negative voltage is applied at the gate of PMOS transistors (NBTI) or a

[1]An address includes – from right to left – *offset* bits identifying the bytes accessed within the cache line, *index* bits identifying the cache set accessed, and *tag* bits that identify different addresses placed in the same cache set.

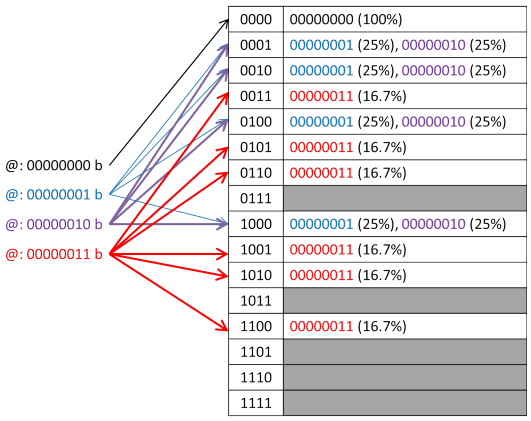

Fig. 3. Example address distribution for a 16-set cache with default random modulo design.

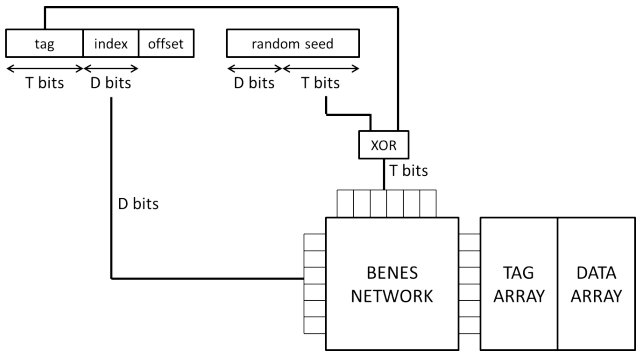

Fig. 4. Schematic of the baseline implementation of a random modulo cache.

positive voltage for NMOS ones (PBTI). This creates new carriers affecting gate delay and leakage as in the case of HCI. However, BTI degradation does not relate to switching activity as HCI does but, instead, relates to the amount of time MOS transistors spend in conductive mode. This effect is particularly relevant for cache memories since they are typically implemented with the smallest devices – so the ones that may fail earlier – and conventional 6T and 8T cells consist of 2 inverters arranged in a ring fashion so that the PMOS transistor of one inverter degrades due to NBTI and the NMOS transistor of the other inverter degrades due to PBTI regardless of the cell contents. In this context, it has been observed that aging is maximized when the cell stores always the same value, so two transistors degrade constantly whereas the other two do not degrade at all. Conversely, aging is minimized when the cell stores a '0' 50% of the time and a '1' also 50% of the time [7]. Therefore, we will study how random placement can mitigate BTI by balancing the contents of cache cells. While different works offer divergent views on the benefits of the self-healing effect for BTI, this has no impact on the conclusions of our work, but on the actual lifetime gains that could be obtained. In particular, some authors show that, whenever transistors are not degrading, their degradation rolls back progressively to some extent [7]. Other authors, however, notice that such recovery rolls back very quickly when the transistors are stressed again, thus questioning to what extent this recovery extends the lifetime of transistors [36]. Nevertheless, the best and worst case for SRAM cells do not change regardless of the benefits of the self-healing effect. Hence, in our particular study we stick to a first-order approximation for BTI effects relating BTI degradation to stress duration, thus disregarding the self-healing effect. As shown later in our evaluation, our conclusions would not be affected by this effect if taken into account.

III. Enhanced Aging-Friendly Random Cache Placement

In this section we first describe the functioning of the default *RM* cache design, used for L1 caches, illustrating its limitations related to aging. We then present our new enhanced *RM* implementation that makes a far more balanced use of the cache sets, thus more friendly from an (mostly HCI) aging perspective. Finally, we study *hRP* set distribution for L2 caches, and show that it uses cache sets in a fairly balanced way.

A. Random Modulo Set Distribution

RM is intended to randomly distribute addresses (indexes in particular) among cache sets. To that end, RM combines address tags with random bits from the random seed to set control signals in the Benes network. The goal is to reach a homogeneous and random permutation selection for index bits. However, index bits themselves are not random since they are completely program dependent.

Let us illustrate this with an example. Let us assume a program accessing 4 addresses (once offset bits have been removed): 0x00 (00000000b), 0x01 (00000001b), 0x02 (00000010b) and 0x03 (00000011b), and a cache memory with 16 sets, so that only the 4 lowermost bits are used for choosing the set being accessed. Note that offset bits indicate the bytes accessed within the cache line, but do not relate to cache line identification, so they are irrelevant for this example.

- Interestingly, regardless of the particular bit permutation selected, address 0x00 can only be placed in set 0 (0000b). Since the 4 lowermost bits are "0", any permutation of them leads to exactly the same set identifier: 0000b.
- Instead, addresses 0x01 and 0x02 can be mapped to any set such that its set identifier contains exactly one "1" given that their binary representation has exactly one "1" and three "0" (0001b and 0010b respectively) in the 4 lowermost bit positions. This corresponds to sets 1 (0001b), 2 (0010b), 4 (0100b) and 8 (1000b). Any other set cannot be reached with bit permutations of addresses 0x01 and 0x02 regardless of the permutation selected since their lowermost 4 bits – those forming the cache index – only contain one "1".
- Finally, address 0x03 can only be mapped to sets 3 (0011b), 5 (0101b), 6 (0110b), 9 (1001b), 10 (1010b) and 12 (1100b) since they are the only ones whose set identifiers contain exactly 2 bits set to "1" (these are the only permutations possible with 0011b as input).

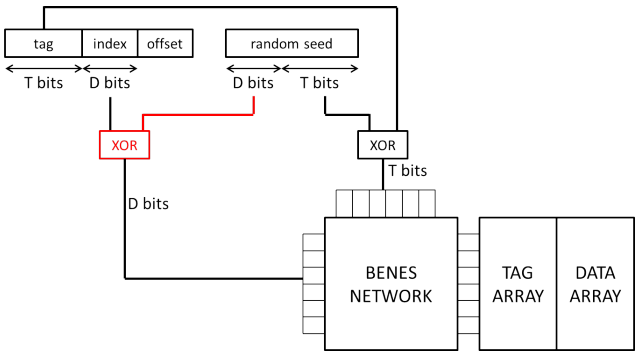

Fig. 5. Schematic of the *enhanced* implementation of a random modulo cache. Red parts indicate the changes introduced.

This is graphically illustrated in Figure 3, where we can see that the frequency with which each set is used is highly heterogeneous. For instance, set 0 is intensively used by address 0x00 (100% of the times address 0x00 is placed in set 0). Some other sets are used with different degrees of intensity by addresses 0x01, 0x02 and 0x03. For instance, addresses 0x01 and 0x02 are mapped to set 1 25% of the times each (permutation 0001b). Some other sets (7, 11, 13, 14, 15) are never used since no address has enough bits set to "1" to be mapped to any of those sets under any permutation. For instance, since no address has 4 bits set to "1", no bit permutation can make any address access set 15. Further note that, if addresses are accessed heterogeneously, the impairment in the use of the different cache sets can be potentially much higher.

Having an heterogeneous cache set utilization is expected to lead to higher degradation for the most used sets due to, for instance, HCI among other sources of transistor degradation, since HCI affects devices proportionally to their switching activity, which in turn depends on the access distribution across cache sets.

In terms of balancing contents stored, relevant for BTI, placement cannot vary what contents are stored in each particular bit of the line. Thus, if a particular bit (e.g., bit 30) is highly biased towards '0', placement functions cannot change this fact. However, when all addresses make an homogeneous use of the sets, the maximum bias of any particular bit in each position is minimized. For instance, if 90% of the lines have bit 30 set to '0', a deterministic placement may make that some specific sets have bit 30 100% of the time set to '0', whereas others have it set to '0' less than 90% of the time. This would cause the fastest degradation for that bit in some sets, making cache fail earlier. Instead, with a perfectly randomized cache placement all cache lines will have 90% bias for bit 30, thus delaying the occurrence of the first failure.

Hence, we aim at finding a better *RM* implementation that balances utilization of the cache sets regardless of the particular access pattern and indexes of the addresses accessed.

B. Randomizing Set Distribution

Our proposal to make index bits have a random distribution is analogous to that used for making control bits in the Benes network be random: hashing address bits with random bits. In the particular case of the index bits, we XOR them with some random bits of the random seed. For instance, let us recall our previous example. If we XOR the index bits of 0x00 (so 0000b) with a random value, in essence we will obtain 16 different indexes – all binary values that can be encoded with 4 bits – with homogeneous probability. This also holds for any other address regardless of their particular index bits. Thus, all addresses are placed to all sets with identical probability regardless of their particular index bits. Therefore, in the long run all sets are expected to be used homogeneously regardless of the particular access patterns of the programs being run.

The drawback of this approach is that a XOR gate is introduced in the path of the index bits to the Benes network, thus potentially affecting the critical path. Still, since a single XOR gate is added, the impact is limited as proven later in the evaluation section.

Figures 4 and 5 show the baseline *RM* design and our enhanced *RM* design respectively. As shown, in our enhanced version we add a level of XOR gates to combine index bits (D bits) with D random bits taken from the random seed. In the figure we show that the random bits used for the index generation (those in the left side of the Benes network) and control bits generation (those on the top part of the Benes network) correspond to different random bits. In practice there is no constraint on using the same bits or different ones since they are used for different purposes.

In summary, as shown later, this conceptually minor – but highly powerful – modification allows balancing the utilization of the cache sets, thus mitigating maximum aging and so increasing the lifetime of the cache memory. The impact in the critical path is low (at most an extra XOR gate), address-to-set mapping within cache way boundaries is a permutation (thus keeping miss rates low by avoiding many potential conflicts), and MBPTA compliance is preserved since cache set location is random.

C. Hash Random Placement Set Distribution

hRP has been designed with the aim of making each address have the same probability to be mapped to each set. In the design of *hRP* in Figure 1 we can observe that the 3 leftmost rotate blocks use as input random bits, which are rotated based on some address bits. The 3 rightmost rotate blocks do the opposite: use as input address bits and rotate them based on some random bits. Overall, the output of the 3 leftmost blocks is a set of random bits, which are later XORed with the other bits, thus leading to a purely random output. Hence, regardless of the input address being accessed, its probability of being mapped to each different cache set is homogeneous in the long run. Therefore, the default *hRP* design already achieves the homogeneous set distribution of addresses for L2 caches obtained with our proposed enhanced *RM* for L1 caches. The homogeneous set distribution of *hRP* has already been proven before [21].

IV. Evaluation

This section evaluates our enhanced *RM* placement and the default *hRP* and their impact on aging. First, we introduce the

evaluation methodology. Then we present the results in terms of access distribution across sets, and how this can improve lifetime in terms of HCI and BTI. Finally, we show the impact of the hardware modification in the critical path.

A. Methodology

We model the first level instruction (IL1) and data (DL1) caches of a NGMP 4-core processor designed for the Space domain [6]. Those caches are 16KB 4-way 32B/line. Thus, they have 128 sets each.

We evaluate the different cache placement designs: modulo (*M*), default Random Modulo (*RM*) and our enhanced random modulo (*ERM*) for L1 caches, and modulo and *hRP* for L2 caches. For our evaluation we use the EEMBC autobench suite, a well-known benchmark suite used in the real-time domain [25]. Each EEMBC benchmark is analyzed using its default input data. Considering multipath effects in the context of MBPTA has been addressed elsewhere [37] and is orthogonal to the work in this paper.

Benchmarks have been run once in an improved version of SoCLib [31] to extract instruction and data address traces. Then, cache set distribution of each placement function has been evaluated in a cache simulator processing those address traces.

For estimating the HCI lifetime improvement, we use the expressions provided in [18] showing that transistors lifetime degradation due to HCI is inversely proportional to their switching activity. For analyzing the potential impact on BTI lifetime, we measure bit bias for all cache bits. Whether the relation of bit bias with lifetime is linear (self-healing effect is negligible) or exponential (self-healing is significant) does not change the conclusions reached analyzing bit bias only.

To quantify delay overheads of the *ERM* implementation we have described both circuit implementations, the original *RM* and the enhanced one, with VHDL and synthesized them using Synopsys DC [1] with a TSMC 45nm technology library [9]. Additionally, both implementations have been integrated in a 4-core Leon3-based processor resembling the NGMP and synthesized in a Stratix IV Altera device at 100MHz.

B. Set distribution

First we evaluate the access distribution across sets for each cache (IL1, DL1 and L2) and placement function: modulo (M), random modulo (RM) and our enhanced random modulo (ERM) for L1 caches, and M and hRP for L2. For each one we show the ratio between the maximum number of accesses per set and the average number of accesses per set (MAX/AVG). In the ideal case where accesses are perfectly balanced, MAX/AVG should be exactly 1. In general we can expect MAX/AVG to be higher than 1.

Table I summarizes the results for all benchmarks for L1 caches. We use $100,000$ runs with different random seeds for RM and ERM. Since M always delivers the same distribution, one run suffices. As shown, M produces high imbalance across sets, particularly for the DL1 cache. The particular addresses accessed determine the sets accessed, and so the set distribution. Thus, M distribution is completely program-dependent. This is in part mitigated for the IL1 since loops contain some significant sequential code accessed many times, thus leading to quite homogeneous distribution (at least for the sets accessed in the loop). Conversely, access patterns for DL1 can be highly irregular in many cases, thus leading to high imbalance in the set access distribution. The harmonic mean for the set distribution of M is 4.9 for the IL1 and 27.9 for the DL1, thus far from the ideal value 1.0.

TABLE I
DISTRIBUTION OF ACCESSES ACROSS SETS FOR THE DIFFERENT PLACEMENT FUNCTIONS AND L1 CACHES.

Cache	IL1			DL1		
Placement	**M**	**RM**	**ERM**	**M**	**RM**	**ERM**
a2time	5.5	1.5	1.01	32.0	3.5	1.03
aifftr	3.2	2.2	1.01	17.3	2.1	1.02
aifirf	5.9	1.4	1.01	27.7	10.9	1.03
aiifft	2.4	1.5	1.01	17.2	2.1	1.02
basefp	7.4	1.8	1.01	36.6	4.1	1.04
bitmnp	2.1	1.4	1.00	26.8	2.6	1.03
cacheb	12.7	2.6	1.02	24.8	2.3	1.04
canrdr	11.1	2.0	1.01	36.4	5.7	1.04
idctrn	3.0	2.9	1.01	26.7	2.8	1.03
iirflt	7.7	1.3	1.01	20.6	4.2	1.02
pntrch	3.9	1.8	1.01	30.9	3.4	1.02
puwmod	19.1	2.7	1.02	63.3	3.2	1.04
rspeed	8.5	2.1	1.01	41.7	3.6	1.03
tblook	4.2	2.0	1.01	22.1	4.2	1.03
ttsprk	18.6	2.1	1.03	53.8	7.1	1.04
HARMEAN	4.9	1.8	1.01	27.9	3.4	1.03

RM balances accesses much better due to the randomness introduced in the generation of the set index. This is particularly noticeable for the DL1. Still, since some dependence exists between the actual addresses accessed and the sets where they map, set distribution improves only to some extent. The harmonic mean for the set distribution of RM is 1.8 for the IL1 and 3.4 for the DL1. While these results are far better than for M, they are still far from the ideal value 1.0, especially for the DL1.

Finally, our ERM removes the dependence of the set index on the particular address accessed, thus delivering much better set access distributions. This effect is particularly relevant for the DL1, where the imbalance for both M and RM is high. The harmonic mean for the set distribution of ERM is 1.01 (Max:1.03) for the IL1 and 1.03 (Max:1.04) for the DL1, very close to the ideal value 1.0. This translates into marginally better average performance (below 0.1% performance improvement).

For completeness, we show the per-set distribution for the different L1 placement policies for 2 specific examples: the IL1 and the DL1 for `pntrch`. The former (see Figure 6) is a relatively good case for M, whereas the latter (see Figure 7) is a relatively bad case for M. Figures show in the x-axis the different cache sets and in the y-axis the utilization normalized w.r.t. the highest utilization in the M case.

The example in Figure 6 shows that M uses some sets quite often, whereas others are barely used. Still, the number of sets used often is relatively large. RM achieves a much better distribution across sets and only some sets have higher utilization than the average. Those sets correspond to those

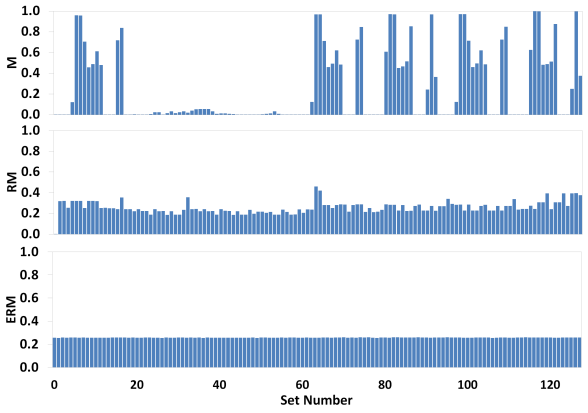

Fig. 6. IL1 per-set access distribution for `pntrch`.

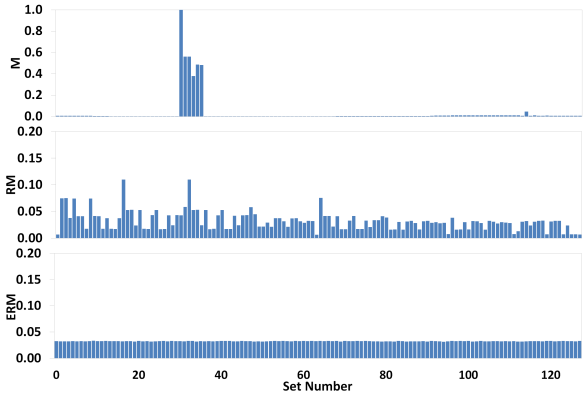

Fig. 7. DL1 per-set access distribution for `pntrch`.

with most index bits being zero (or one), so that randomization has limited effect. Finally, our ERM achieves almost homogeneous cache set utilization.

The example in Figure 7, instead, shows that M uses very few sets of the DL1 cache for `pntrch`. This leads to an extremely unbalanced distribution. In the case of RM (note that the y-axis only reaches 0.2) the distribution is far better as the most used set is used around 8 times less than for M. Still, unbalance is high. Finally, ERM achieves almost homogeneous set utilization.

When analyzing set distribution for the L2 cache (see Table II), we observe that the set distribution for M is highly biased, ranging between 99.7 and 866.2 across benchmarks, being much worse than for L1 caches. This occurs due to two combined effects: (1) the number of sets in L2 is much larger, thus increasing the chances that highly used sets are still highly used whereas many more sets remain mostly unused; and (2) L1 caches filter many accesses to L2, which typically increases the imbalance across sets. hRP instead achieves a highly homogeneous distribution across sets with an harmonic mean of 1.15 across benchmarks and a maximum of 1.23 for `canrdr` benchmark. Moreover, hRP imbalance progressively approaches 1.0 as we increase the number of runs, which is the expected value after an infinite number of runs due to the purely random address-to-set mapping nature of hRP.

TABLE II
DISTRIBUTION OF ACCESSES ACROSS SETS FOR THE DIFFERENT PLACEMENT FUNCTIONS IN THE L2 CACHE.

Cache	L2	
Placement	M	hRP
a2time	679.9	1.17
aifftr	160.3	1.08
aifirf	396.7	1.17
aiifft	159.9	1.08
basefp	712.6	1.16
bitmnp	99.7	1.09
cacheb	353.1	1.15
canrdr	772.8	1.23
idctrn	631.3	1.18
iirflt	583.9	1.16
pntrch	294.8	1.12
puwmod	866.2	1.19
rspeed	638.1	1.17
tblook	651.6	1.18
ttsprk	829.8	1.20
HARMEAN	339.4	1.15

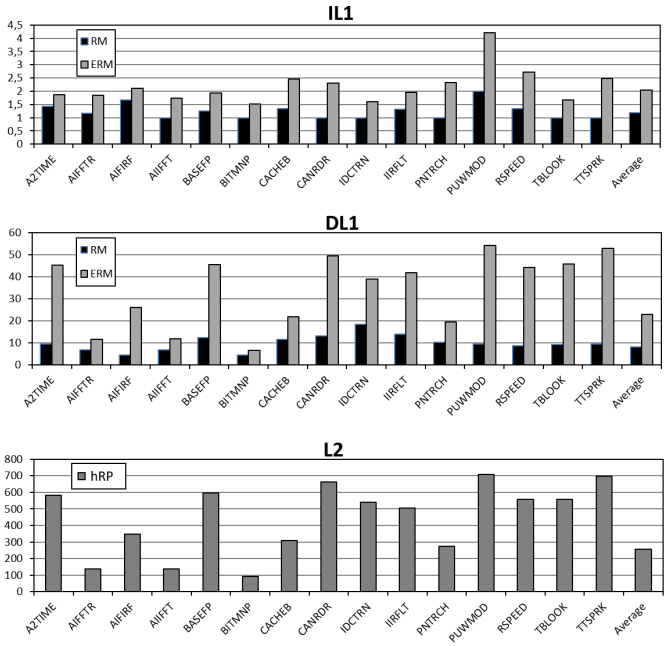

Fig. 8. HCI lifetime for RM and ERM normalized w.r.t. M for both IL1 and DL1, and for hRP normalized w.r.t. M for L2. Scales are different in each plot.

C. Lifetime

Next we elaborate on the impact of the improved set distribution on HCI and BTI lifetime.

HCI. We use the HCI model reported in [18], as explained before, where it is shown that HCI is directly proportional to the switching activity. Thus, we use actual switching activity of the cache accesses to estimate the relative HCI lifetime improvements. We assume that a failure occurs when the first permanent fault due to HCI occurs. Thus, the bit with highest switching activity determines cache lifetime. While other models could be used, our work is centered around safety-related real-time systems where timing verification occurs before operation and assuming that the processor is fault-

free. Thus, unless otherwise considered during the analysis phase, one faulty cache line may impact the WCET and thus, invalidates those timing guarantees on which the certification process has been conducted.

Since the actual lifetime value depends not only on HCI, but also on other sources of failure and hardware components, we report how much the lifetime of the IL1 and DL1 is extended due to HCI for RM and ERM normalizing the results w.r.t. M placement. Results are shown in Figure 8. We observe that lifetime grows by 1.2x and 8.2x on average for IL1 and DL1 respectively for RM. Results for ERM are far better, extending HCI lifetime by 2.1x and 22.8x on average for IL1 and DL1 respectively. If we compare ERM w.r.t. RM, lifetime grows by a factor of 1.7x and 2.8x for IL1 and DL1 respectively.

Regarding the L2 cache, we observe that hRP extends L2 lifetime by a factor of 257x w.r.t. M placement, which is a huge gain. Still, one must consider that HCI is proportional to switching activity and L2 cache lines are typically read/written only a fraction of the times L1 caches are. Therefore, if L1 and L2 caches use the same type of transistors, L1 caches experience much higher switching activity and thus, L1 caches are the reliability bottleneck for HCI. If, instead, L2 caches are implemented with smaller transistors for achieving further integration, then L2 caches may be the reliability bottleneck for HCI, particularly if DL1 is write-through so that all store operations are forwarded to L2 cache. Note that write-through DL1 caches are common in the safety-related domain due to the complexity to implement complex error correcting codes in DL1 caches without impacting cycle time or cache latency [6]. Instead, these latencies are better tolerated in L2 caches and L1 caches can implement parity instead since error correction can be performed using L2 cache contents.

BTI. As explained before, BTI aging does not relate to the switching activity but to the content bias of the cache cells. In order to understand the benefits of ERM in L1 caches and hRP in the L2 cache, we have performed the following experiment for the IL1: we collect the bit bias for each cache line bit considering the actual switching activity for the EEMBC benchmarks and disregarding the effect of replacement policies, therefore using a 1-way IL1. We have performed this experiment for all benchmarks and obtained similar results across all of them. While this experiment provides only a first-order approximation to the impact of ERM on BTI, our conclusions would hold with any BTI lifetime model in the light of the results presented next. Figure 9 shows in the top part the bit bias obtained for each bit of each set in the IL1 for M placement for `pntrch` EEMBC benchmark, where lighter bits indicate they are highly biased towards '1' whereas darker bits are highly biased towards '0'. As shown, many bits are very highly biased towards specific values. In fact, 235 out of the 256 bits per cache line are 100% biased towards '0' or towards '1' in at least one of 128 cache sets. Thus, independently of how much time each cache line is stored in cache and how the replacement policy distributes cache lines across ways, there will be some bits 100% biased towards '0' or '1', thus experiencing maximum degradation. Note that the distribution is heterogeneous across sets, thus reflecting the fact that the particular sets used with M placement directly depend on the memory addresses where objects (code in this case) are stored.

We have evaluated what ERM could achieve in terms of bit bias considering actual switching activity, also disregarding the effect of replacement policies as for M placement. The bit map, again for `pntrch`, is shown in the bottom part of Figure 9. As shown, ERM homogenizes the bias across sets (visually across rows), but cannot do anything to further reduce the bias. This would bring some gains in terms of BTI lifetime since the maximum bias across all bits in cache decreases from 100% to 80%, which would extend lifetime. How much lifetime would be extended would depend on the actual technology and the degree of self-healing it achieves. Lifetime gains would be proportional to the maximum bit bias reduction if self-healing is neglected. Considering self-healing would affect those results, which could be potentially optimistic. In any case, the bit bias achieved is still far away from the ideal 50%. Similar conclusions are reached for ERM in DL1 and hRP in L2 cache across all benchmarks since bias in those caches reaches 100% for M, but remains far away from the 50% for randomized designs. Hence, caches implementing some form of random placement are expected to achieve higher BTI lifetime than those implementing modulo placement based on our bit bias measurements, but are far from the lifetime gains that could be achieved by fully balancing bit bias with techniques such as those in [10], [23], [2] based on inverting cache contents. This effect is expected since random placement homogeneizes bit bias across cache sets, but cannot do anything if specific bit positions are highly biased towards one value.

Therefore, we can conclude that, while randomized caches could provide some benefits in terms of BTI lifetime, other orthogonal techniques based on content inverting are likely needed to mitigate BTI aging if it is a reliability bottleneck.

D. Delay impact

We have synthesized our ERM design using a TSMC 45nm technology library to measure its impact on cycle time. After synthesis we have seen ERM has zero impact w.r.t. the maximum operating frequency that the regular RM implementation can achieve. RM critical path depends on the complexity introduced for XORing and combining TAG bits and random seed bits to configure the Benes network. With the implementation described in [15] the critical path for both ERM and RM is 1.01ns whereas the path affected by the inclusion of the XOR gate has 0.22ns delay in the case of ERM (0.09ns for RM plus 0.13ns due to the level of XOR gates added). While faster RM implementations could be devised, they require at least two levels of XOR gates to combine TAG and random bits, thus being 0.26ns the lowest potential delay (still higher than 0.22ns).

V. Related Work

WCET estimation has been an important concern for both academia and industry during many years [35], [5]. MBPTA [8] has been devised to obtain trustworthy and tight

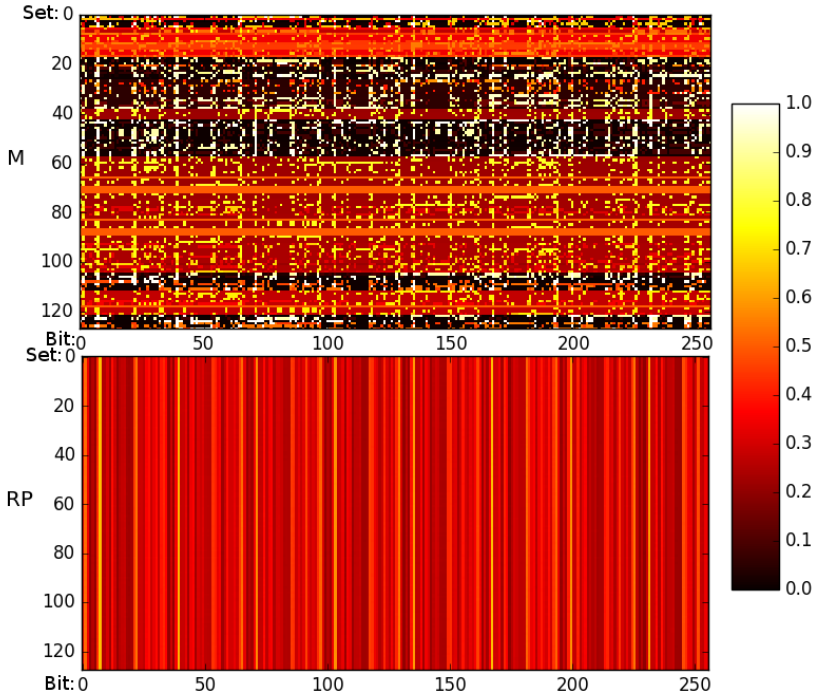

Fig. 9. Bit bias for the IL1 cache with M (top) and ERM (bottom) placement policies for the `pntrch` EEMBC benchmark.

WCET estimates for complex software running on complex hardware where end users can only afford using measurement-based techniques. MBPTA has been successfully assessed with some industrial case studies [33], [34].

Several approaches have been proposed in the literature to address the impact of HCI and BTI in processors and cache structures [2], [28], [23]. While the majority of works focus on BTI effects, some recent works have also pointed out the importance of considering the effects of HCI degradation [16], [18], [10]. In fact, authors in [10] have already proposed improving the uniformity of accesses to the cache to mitigate HCI and NBTI degradation. Unlike in [10], where authors rely on introducing dedicated hardware resources to achieve uniform access distribution, we rely on the good properties of random modulo [15] and hash-based random placement [19], [20], [21] to achieve the same goal at roughly no cost.

Considering WCET estimation together with reliability issues has been done at several fronts. Some authors propose preserving WCET estimation methods by devising hardware cache designs able to tolerate permanent faults with no effect (or easy to account effect) on WCET estimates [3], [4]. Other authors propose accounting for the timing impact of faults in a probabilistic manner in combination with static deterministic timing analysis methods by studying the impact and probabilities of different fault distributions [11], [12], [13]. However, since those approaches do not use randomized cache designs, the impact estimation of (random) faults has to rely on the most conservative assumptions. Additionally, they cannot be applied in the context of measurement-based timing analysis.

Recently, some authors have done some preliminary work in the context of WCET analysis of faulty hardware together with MBPTA [29], [30]. Results are promising and prove that the random nature of the timing of hardware providing the properties required by MBPTA matches very well with the random nature of faults, thus leading to efficient solutions. While that work shows to be efficient to account for the impact of transient faults and a limited number of permanent faults, it does not provide means to mitigate aging effects. In this paper, instead, we assess the reliability of the different random placement designs, random modulo for L1 caches and hash-based random placement for L2 caches, in terms of aging (HCI and BTI). Then, we propose an enhanced random modulo implementation such that reliability of L1 caches is improved while preserving the good properties of random modulo.

VI. Conclusions

Fault tolerance and WCET estimation, needed both for safety-related real-time systems verification, have often been addressed as separate concerns. In this context, approaches based on MBPTA have been shown to match very well the needs of both concerns by relying on the same principle: randomness. Therefore, efficient solutions can be built to consider both concerns simultaneously.

In this paper we assess the reliability in terms of HCI and BTI of random modulo cache designs for L1 caches and hash-based random placement for L2 caches, proven convenient for MBPTA. Our results show that random placement designs effectively mitigate HCI aging and provide some – limited – benefits in terms of BTI. Still, random modulo is far from being optimal in terms of HCI. Therefore, we propose an alternative random modulo implementation that further mitigates HCI aging while preserving the main features of random modulo: low impact in critical path, low miss rates and adherence to the requirements of MBPTA.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROXIMA Project (www.proxima-project.eu), grant agreement no 611085. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness and FEDER funds through grant TIN2014-60404-JIN.

REFERENCES

[1] Synopsys design compiler. Technical report.
[2] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-aware processor. In *MICRO*, pages 85–96, 2007.
[3] J. Abella et al. RVC: A mechanism for time-analyzable real-time processors with faulty caches. In *HiPEAC Conference*, 2011.
[4] J. Abella et al. RVC-Based time-predictable faulty caches for safety-critical systems. In *IOLTS*, 2011.
[5] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.
[6] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.
[7] M. A. Alam. A critical examination of the mechanics of dynamic NBTI for PMOSFETs. In *International Electron Devices Meeting (IEDM)*, pages 14.4.1–14.4.4, 2003.
[8] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
[9] TSMC foundry. TSMC 40 nm technology.
[10] E. Gunadi et al. Combating aging with the colt duty cycle equalizer. In *MICRO*, 2010.
[11] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *RTNS*, 2013.
[12] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. *Journal of Real-Time Systems*, 51(2):128–152, 2015.
[13] D. Hardy et al. Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults. In *DATE*, 2016.
[14] C. Hernandez et al. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *DASIA*, 2015.
[15] C. Hernandez et al. Random modulo: a new processor cache design for real-time critical systems. In *DAC*, 2016.
[16] V. Huard et al. Managing sram reliability from bitcell to library level. In *IRPS*, 2010.
[17] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
[18] H. Kim et al. Use it or lose it: Proactive, deterministic longevity in future chip multiprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 20(4):65:1–65:26, September 2015.
[19] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[20] L. Kosmidis et al. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.
[21] L. Kosmidis et al. Efficient cache designs for probabilistically analysable real-time systems. *IEEE Transactions on Computers*, 63(12):2998–3011, 2014.
[22] L. Kosmidis et al. Probabilistic timing analysis and its impact on processor architecture. In *DSD*, 2014.
[23] S. V. Kumar, K. H. Kim, and S. S. Sapatnekar. Impact of NBTI on SRAM read stability and design for reliability. In *ISQED*, pages 6 pp.–218, 2006.
[24] E. Mezzetti et al. Randomized caches can be pretty useful to hard real-time systems. *LITES*, 2(1), 2015.
[25] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[26] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.
[27] D.K. Schroder and J.A. Babcock. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of Applied Physics*, 94(1):1–18, 2003.
[28] J. Shin et al. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. In *ISCA*, 2008.
[29] M. Slijepcevic et al. DTM: Degraded test mode for fault-aware probabilistic timing analysis. In *ECRTS*, 2013.
[30] M. Slijepcevic et al. Timing verification of fault-tolerant chips for safety-critical applications in harsh environments. *IEEE Micro - Special Series on Harsh Chips*, 34(6), 2014.
[31] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.
[32] C.T. Wang. *Hot Carrier Design Considerations for MOS Devices and Circuits*. Van Nostrand Reinhold, 1990.
[33] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.
[34] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *DATE*, 2015.
[35] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, 2008.
[36] J. Zhang et al. New findings of NBTI in partially depleted SOI transistors with ultra-thin gate dielectrics. In *IRPS*, pages 687–688, 2004.
[37] M. Ziccardi et al. EPC: Extended path coverage for measurement-based probabilistic timing analysis. In *RTSS*, 2015.

David Trilla is a PhD. Student for the CAOS group at BSC. He obtained his M.S. degree in 2016 and graduated in Informatics Engineering in 2014, both titles obtained from the Universitat Politecnica de Catalunya. He enrolled BSC in 2014 and has participated in the European project ESA-HAIR. He has been working on timing prediction models for real-time software for multicore during early design stages and his current research focuses on the effects on energy consumption behavior on randomized architectures.

Jaume Abella is a senior PhD. Researcher at BSC and HiPEAC member. He received his MS (2002) and PhD. (2005) degrees from the UPC. He worked at the Intel Barcelona Research Center (2005-2009) in microarchitectures for fault-tolerance and low power, and memory hierarchies. He joined the BSC in 2009 where he is in charge of hardware designs for FP7 PROXIMA, and BSC tasks in H2020 SAFURE. Jaume is also involved in ESA-BSC bilateral projects. He has authored more than 15 patents and 100 papers in top conferences and journals. He is (has been) co-advisor of ten MS and PhD students.

Carles Hernandez received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politecnica de Valencia, in 2006 and 2012, respectively. He is currently senior PhD. Researcher at the Barcelona Supercomputing Center. His area of expertise includes network-on chip and reliability-aware processor design. He participates (has participated) in NaNoC, parMERASA, PROXIMA IP7 and VeTeSS ARTEMIS projects. In the context of PROXIMA he is in charge of probabilistic technology developments in the Leon3 processor. In 2012 he was intern at Intel Mobile Communications Munich.

Francisco J. Cazorla is the leader of the CAOS group at BSC and member of HIPEAC Network of Excellence. He has led projects funded by industry (IBM and Sun Microsystems), by the European Space Agency (ESA) and public-funded projects (FP7 PROARTIS project and FP7 PROXIMA project). He has participated in FP6 (SARC) and FP7 Projects (MERASA, VeTeSS, parMERASA). His research area focuses on multithreaded for both high-performance and real-time systems on which he is co-advising several PhD theses. He has co-authored 3 patents and over 100 papers in international refereed conferences and journals.